**11 вопрос. Антисимметрия данных/объектов**

Листинг 6.1. Конкретная реализация Point

public class Point {

public double x;

public double y;

}

Листинг 6.2. Абстрактная реализация Point

public interface Point {

double getX();

double getY();

void setCartesian(double x, double y);

double getR();

double getTheta();

void setPolar(double r, double theta);

}

Два предыдущих примера показывают, чем объекты отличаются от структур данных. **Объекты скрывают свои данные за абстракциями и предоставляют функции, работающие с этими данными. Структуры данных раскрывают свои данные и не имеют осмысленных функций.** А теперь еще раз перечитайте эти определения. Обратите внимание на то, как они дополняют друг друга, фактически являясь противоположностями. Различия могут показаться тривиальными, но они приводят к далеко идущим последствиям. Возьмем процедурный пример из листинга 6.5. Класс Geometry работает с тремя классами геометрических фигур. Классы фигур представляют собой простые структуры данных, лишенные какого-либо поведения. Все поведение сосредоточено в классе Geometry.

Листинг 6.5. Процедурные фигуры

public class Square {

public Point topLeft;

public double side;

}

public class Rectangle {

public Point topLeft;

public double height;

public double width;

}

public class Circle {

public Point center;

public double radius;

}

public class Geometry {

public final double PI = 3.141592653589793;

public double area(Object shape) throws NoSuchShapeException {

if (shape instanceof Square) {

Square s = (Square)shape;

return s.side \* s.side;

} else if (shape instanceof Rectangle) {

Rectangle r = (Rectangle)shape;

return r.height \* r.width;

} else if (shape instanceof Circle) {

Circle c = (Circle)shape;

return PI \* c.radius \* c.radius;

}

throw new NoSuchShapeException();

}

}

Объектно-ориентированный программист недовольно поморщится и пожалуется на процедурную природу реализации — и будет прав. Но возможно, его презрительная усмешка не обоснована. Подумайте, что произойдет при включении в Geometry функции perimeter(). Классы фигур остаются неизменными! И все остальные классы, зависящие от них, тоже остаются неизменными! С другой стороны, при добавлении новой разновидности фигур мне придется изменять все функции Geometry, чтобы они могли работать с ней. Перечитайте еще раз. Обратите внимание на то, что эти два условия диаметрально противоположны.

Теперь рассмотрим объектно-ориентированное решение из листинга 6.6. Метод area() является полиморфным, класс Geometry становится лишним. Добавление новой фигуры не затрагивает ни одну из существующих функций, но при добавлении новой функции приходится изменять все фигуры!

Листинг 6.6. Полиморфные фигуры

public class Square implements Shape {

private Point topLeft;

private double side;

public double area() {

return side\*side;

}

}

public class Rectangle implements Shape {

private Point topLeft;

private double height;

private double width;

public double area() {

return height \* width;

}

}

public class Circle implements Shape {

private Point center;

private double radius;

public final double PI = 3.141592653589793;

public double area() {

return PI \* radius \* radius;

}

}

И снова мы наблюдаем взаимодополняющую природу этих двух определений. В этом проявляется основополагающая дихотомия между объектами и структурами данных.

**Процедурный код (код, использующий структуры данных) позволяет легко добавлять новые функции без изменения существующих структур данных. Объектно-ориентированный код, напротив, упрощает добавление новых классов без изменения существующих функций.**

Обратные утверждения также истинны.

**Процедурный код усложняет добавление новых структур данных, потому что оно требует изменения всех функций. Объектно-ориентированный код усложняет добавление новых функций, потому что для этого должны измениться все классы.**

**Таким образом, то, что сложно в ОО, просто в процедурном программировании, а то, что сложно в процедурном программировании, просто в ОО!**

В любой сложной системе возникают ситуации, когда вместо новых функций в систему требуется включить новые типы данных. Для таких ситуаций объекты и объектно-ориентированное программирование особенно уместны. Впрочем, бывает и обратное — вместо новых типов данных требуется добавить новые функции. Тогда лучше подходит процедурный код и структуры данных.